## 

[Структуры данных](#_41n2am3ndnd8)

[Массивы](#_gwtj0tenvos0)

[Слайсы](#_8uurb671e91r)

[Ветвления и циклы](#_k5u8vy1ulrvc)

[Методы и интерфейсы](#_z0b0vl3spwou)

[Конкурентный доступ](#_coq1m0tkjm1p)

[defer](#_2j79tlw8avgn)

# Структуры данных

Объявление локальной переменной типа string

**var** stringVariable1 string

Присваивание значения переменной My

stringVariable1 = **"Direct string"**

println(stringVariable1)

Объявить новую переменную и присвоить значение

stringVariable2 := **"String variable 1"**

println(stringVariable2)

Объявленной переменной можно только присваивать, внутри своей области видимости переменных (scope)

stringVariable2 = **"String variable 2"**

println(stringVariable2)

## Массивы

array := [...]string{**"element1"**, **"element2"**, **"element3"**}

println(array[0])

## Слайсы

Слайс - это обертка над массивом с дополнительным размером Capacity, определяет максимальный объем в памяти. В основном в языке Go используются слайсы, а не массивы.

slice := [...]string{**"element1"**, **"element2"**, **"element3"**}

println(slice[1])

Структура для хранения данных в виде пар, ключ - значение

**var** timeZone = **map**[string]int{

**"UTC"**: 0\*60\*60,

**"EST"**: -5\*60\*60,

**"CST"**: -6\*60\*60,

**"MST"**: -7\*60\*60,

**"PST"**: -8\*60\*60,

}

println(timeZone[**"MST"**])

Оператор := возвращает указатель, адрес в памяти на структуру GoStruct

goStruct := new(GoStruct)

println((\*goStruct).Field1)

goStruct.Field1 = **"field 1"**

println(goStruct.Field1)

Возвращает значение структуры GoStruct

goStructVar := GoStruct{**"field 1"**, **"field 2"**}

println(goStructVar.Field2)

# Ветвления и циклы

if вида: if инициализация; условие

**if** f, err := os.Open(**"some file's name"**); err != nil {

log.Print(**"Error! File not found!"**)

} **else** {

fmt.Printf(f.Name())

}

Как цикл for в C

for инициализация; условие; действие после проверки условия { }

sum := 0

**for** i := 0; i < 10; i++ {

sum += i

}

array := [...]string{**"string1"**, **"string2"**, **"string3"**}

Как цикл while в C

for условие { }

**for** element := **range** array {

fmt.Printf(array[element])

}

Как цикл while(true) в C#

for { }

**for** {

**break**

}

# Методы и интерфейсы

**type** GoStruct **struct** {

Field1 string

Field2 string

}

Функция может принадлежать структуре

**func** (m GoStruct) TwoFiels() string {

**return** m.Field1 + **" "** + m.Field2

}

**func** TwoFiedls(field1 string, field2 string) string {

**return** field1 + **" "** + field2

}

**type** TwoFieldsConcatenater **interface** {

TwoFields() string

}

goStructVar := GoStruct{**"field 1"**, **"field 2"**}

Структура может содержать методы

twoFields := goStructVar.TwoFiels()

println(twoFields)

println(TwoFiedls(goStructVar.Field1, goStructVar.Field2))

Объявление ссылки на интерфейс

**var** interfaceToStruct TwoFieldsConcatenater

Получить ссылку как адрес переменной goStructVar

interfaceToStruct = &goStructVar

Через ссылку на интерфейс можно вызвать его метод

println(interfaceToStruct.TwoFields());

# Конкурентный доступ

**func** say(s **chan** string) {

str := **"world"**

time.Sleep(1000 \* time.***Millisecond***)

s <- str

}

msgChan := make(**chan** string, 100)

Запустить в другом goroutine - поток, управляемый Go runtime

**go** say(msgChan)

println(**"hello"**)

println(msgChan)

Получить сообщение из канала msgChan и присвоить значение в переменую msgString

msgString := <- msgChan

println(msgString)

## defer

Отложенное выполнение функции. Выполняется после исполнения return

**func** trace(s string) {

fmt.Println(**"entering:"**, s)

}

**func** retstring() string {

**return "string"**

}

**func** untrace(s string) {

fmt.Println(**"leaving:"**, s)

}

**func** a() string {

**defer** untrace(**"a"**)

trace(**"a"**)

**return** retstring()

}

str := a()

println(str)